



Міністерство освіти і науки України  
Національний технічний університет України  
“Київський політехнічний інститут імені Ігоря Сікорського”  
Факультет інформатики та обчислювальної техніки  
Кафедра інформаційних систем та технологій

Лабораторна робота №4  
**Технології розробки програмного забезпечення**  
«ШАБЛОНИ «SINGLETON», «ITERATOR», «PROXY»,  
«STATE», «STRATEGY»»

Виконав:  
студент групи ІА-24  
Красношاپка Р. О.  
Перевірів:  
Мягкий М. Ю.

## Зміст

Короткі теоретичні відомості.....	3
Хід роботи .....	4
Реалізація шаблону проєктування для майбутньої системи .....	4
Зображення структури шаблону .....	8
Посилання на репозиторій.....	8
Висновок.....	8

## **Короткі теоретичні відомості**

**Singleton (Одинак)** – це патерн проектування, який забезпечує наявність лише одного екземпляра класу та надає глобальну точку доступу до нього. Використовується для централізованого керування доступом до певних ресурсів, таких як налаштування чи логування. У багатопоточному середовищі важливо забезпечити потокобезпеку при його реалізації.

**Iterator (Ітератор)** – це поведінковий патерн, що забезпечує зручний спосіб послідовного доступу до елементів колекції без розкриття її внутрішньої реалізації. Дозволяє обходити елементи у різних напрямках (наприклад, вперед чи назад), приховуючи складнощі структури даних.

**Proxy (Замісник)** – це структурний патерн, який створює об'єкт-замісник для контролю доступу до іншого об'єкта. Використовується для оптимізації роботи, наприклад, шляхом відкладеної ініціалізації, контролю доступу, кешування чи авторизації. Замісник додає додаткову логіку перед викликом основного об'єкта.

**State (Стан)** – це поведінковий патерн, що дозволяє об'єкту змінювати свою поведінку залежно від його стану. Логіка розділяється між окремими класами станів, що полегшує розширення та підтримку. Ключова особливість полягає у зміні станів під час виконання, що дає змогу легко керувати переходами між ними.

**Strategy (Стратегія)** – це поведінковий патерн, який визначає набір алгоритмів, інкапсулює кожен з них і дозволяє використовувати їх взаємозамінно. Це дає змогу змінювати алгоритм виконання операції без зміни коду клієнта. Шаблон забезпечує гнучкість архітектури, дозволяючи динамічно обирати стратегію під час виконання програми.

## Хід роботи

**Project Management software (proxy, chain of responsibility, abstract factory, bridge, flyweight, client-server)** Програмне забезпечення для управління проектами повинно мати наступні функції: супровід завдань/вимог/проектів, списків команд, поточних завдань, планування за методологіями agile/kanban/rup (включаючи дошку завдань, ітерації тощо), мати можливість прикріплювати вкладені файли до завдань та посилатися на конкретні версії програми, зберігати виконувані файли для кожної версії.

### Реалізація шаблону проєктування для майбутньої системи

Шаблон **Proxy (Замісник)** є одним із структурних шаблонів проєктування. Його основна мета – створити об'єкт-замісник, який контролює доступ до реального об'єкта, додаючи додаткову функціональність або обмеження. Замісник виступає посередником між клієнтом і реальним об'єктом, зберігаючи ту саму інтерфейсну структуру, що й реальний об'єкт.

### Основні принципи та застосування Proxy:

1. **Контроль доступу:** Proxy дозволяє додати механізми аутентифікації чи авторизації перед тим, як клієнт отримає доступ до основного об'єкта.
2. **Оптимізація ресурсів:** Реальний об'єкт може бути створений лише тоді, коли це дійсно необхідно (наприклад, у випадку відкладеної ініціалізації).
3. **Логування та моніторинг:** Proxy може вести журнал викликів методів або здійснювати моніторинг запитів.
4. **Кешування:** Замісник може зберігати проміжні дані, щоб уникнути надмірних викликів до реального об'єкта.

Проксі-клас `TeamProxyImpl` використовується для перевірки доступу користувача до командних ресурсів. Перед викликом основних методів сервісу, проксі-клас перевіряє, чи користувач є членом команди. Якщо доступ дозволено,

виклик перенаправляється до реального сервісу (TeamServiceImpl), інакше – викидається виняток. Таким чином, Проху додає рівень безпеки та спрощує основну логіку сервісу, залишаючи перевірку доступу за замісником.

```
14 @Service
15 @Slf4j
16 @AllArgsConstructor
17 public class TeamProxyImpl implements TeamProxy {
18     private final UserService userService;
19     private final TeamService teamService;
20
21     4 usages ± QUIRINO
22     private void checkAccess(String teamId, String token) throws AccessDeniedException {
23         User user = userService.getUserByToken(token);
24         if (!teamService.isUserInTeam(teamId, user.getId())) {
25             throw new AccessDeniedException("Access denied: User is not a member of the team");
26         }
27     }
28
29     1 usage ± QUIRINO
30     @Override
31     public TeamDto createTeam(TeamDto teamDto) {
32         return teamService.createTeam(teamDto);
33     }
34
35     1 usage ± QUIRINO
36     @Override
37     public Optional<TeamDto> getTeamById(String id, String token) throws AccessDeniedException {
38         checkAccess(id, token);
39         return teamService.getTeamById(id);
40     }
41 }
```

Рис. 1 – Код класу TeamProxyImpl

Контролер використовує проксі-сервіс для виконання операцій з груповими списками покупок.

```

16  @RestController
17  @RequestMapping("/teams")
18  @AllArgsConstructor
19  @Slf4j
20  public class TeamController {
21
22      private final TeamProxy teamProxy;
23
24      ± QUIRINO ±
25      @PostMapping("/create")
26      public ResponseEntity<Object> createTeam(@RequestBody TeamDto teamDto, @RequestHeader("Authorization") String token) {
27          return handleRequest() -> {
28              TeamDto createdTeam = teamProxy.createTeam(teamDto);
29              return ResponseEntity.ok(Map.of( k1: "status", v1: "success", k2: "data", createdTeam));
30          };
31      }
32
33      ± QUIRINO ±
34      @GetMapping("/get/{id}")
35      public ResponseEntity<Map<String, Object>> getTeamById(@PathVariable String id, @RequestHeader("Authorization") String token) {
36          return handleRequest() -> {
37              Optional<TeamDto> teamDto = teamProxy.getTeamById(id, token);
38              return teamDto.map(dto -> ResponseEntity.ok(Map.of( k1: "status", v1: "success", k2: "data", dto)))
39                  .orElseGet(() -> ResponseEntity.status(HttpStatus.NOT_FOUND).body(Map.of( k1: "status", v1: "error", k2: "message", v2: "T
40      });

```

Рис. 2 – Код класу TeamController

Проксі-клас для TeamController перевіряє, чи є користувач членом команди перед тим, як передати виклик до реального сервісу (TeamServiceImpl). Якщо перевірка успішна, запит передається до реального сервісу. У протилежному випадку викликається виняток, і метод не виконується. Після успішної передачі запиту реальний сервіс обробляє запит і повертає результат через проксі.

```

14  @Service
15  @Slf4j
16  @AllArgsConstructor
17  public class TeamServiceImpl implements TeamService {
18
19      private final TeamRepository teamRepository;
20      private final TeamValidator teamValidator;
21      private final TeamConverter teamConverter;
22
23      1 usage  ± QUIRINO
24      @Override
25      public TeamDto createTeam(TeamDto teamDto) {
26          if (!teamValidator.isValidTeam(id: null, teamDto)) {
27              return null;
28          }
29          Team team = saveTeam(teamConverter.buildTeamFromDto(teamDto));
30          return teamConverter.buildTeamDtoFromTeam(team);
31      }
32
33      1 usage  ± QUIRINO
34      @Override
35      public TeamDto updateTeam(String id, TeamDto teamDto) {
36          if (!teamValidator.isValidTeam(id, teamDto)) {
37              return null;
38          }
39          Team team = teamConverter.buildTeamFromDto(teamDto);
40          team.setTeamId(id);
41          saveTeam(team);
42          return teamConverter.buildTeamDtoFromTeam(team);
43      }

```

Рис. 3 – Код класу TeamServiceImpl

## Зображення структури шаблону

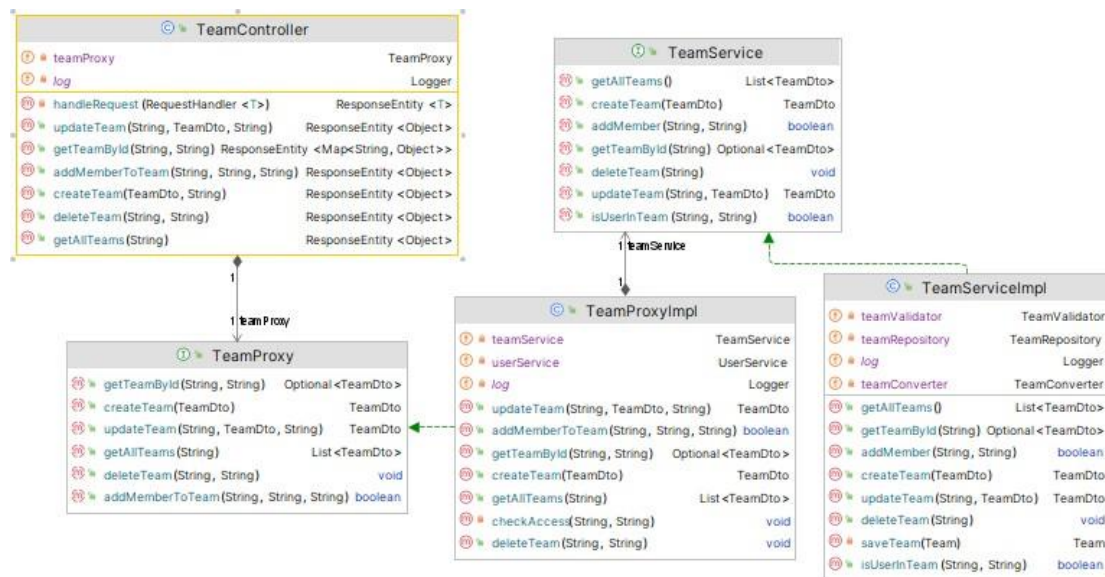


Рис. 4 – Структура шаблону

### Посилання на репозиторій:

<https://github.com/QUIRINO228/projectManagmentSoftware>

**Висновок:** У межах цієї лабораторної роботи було проаналізовано структуру та призначення п'яти ключових шаблонів проектування: Singleton, Iterator, Proxy, State і Strategy. Кожен із цих патернів має свої переваги та обмеження, а їх правильне застосування дозволяє значно підвищити ефективність процесу розробки програмного забезпечення.

У ході роботи детально вивчено призначення кожного шаблону, їх сильні сторони та недоліки. На основі отриманих знань було реалізовано проксі-клас **TeamProxy**, що використовує шаблон Проксу. Це рішення забезпечило перевірку прав доступу (чи є користувач членом групи) перед виконанням методів реального сервісу. Використання шаблону Проксу дозволило чітко розділити відповідальність між перевіркою доступу та основною бізнес-логікою, що позитивно вплинуло на підтримуваність і масштабованість коду.